C# 프로그래밍

- 10 주차 (2강)

SEJONG UNIVERSITY

인터페이스의 프로퍼티
추상 클래스
파일 입출력

SEJONG UNIVERSITY

인터페이스의 프로퍼티

- 인터페이스는 메소드 뿐 아니라 프로퍼티와 인덱서도 가질 수 있음
  - ✓ 파생 클래스에서 프로퍼티와 인덱서를 반드시 구현해야 함
  - ✓ 인터페이스의 프로퍼티 선언은 클래스의 자동구현 프로퍼티 선언과 동일

사용형식:

```
interface 인터페이스이름
{
    public 형식 프로퍼티이름1
    {
        get;set;
    }

    public 형식 프로퍼티이름2
    {
        get; set;
    }
    // ...
}
```

사용예제

```
interface IProduct
{
    string ProductName {
        get; set;
    }
}

class Product : IProduct
{
    private string productName;
    public string ProductName {
        get { return productName; }
        set { productName = value; }
    }
}
```

인터페이스의 프로퍼티 예제 코드

```
interface INamedValue
{
    string Name { get; set; }
    string Value { get; set; }
}

class NamedValue : INamedValue
{// INamedValue 인터페이스 프로퍼티 구현
 // 위해 자동구현 프로퍼티 이용 가능
    public string Name { get; set; }
    public string Value { get; set; }
}
```

```
static void Main(string[] args)
{
    NamedValue name = new NamedValue()
    { Name = "이름", Value = "홍길동" };

    NamedValue height = new NamedValue()
    { Name = "키", Value = "175cm" };

    Console.WriteLine($"{name.Name} : {name.Value}");
    Console.WriteLine($"{height.Name} : {height.Value}");
}
```

출력 결과:
이름 : 홍길동
키 : 175cm

추상 클래스: 인터페이스와 클래스 사이

- 추상 클래스는 abstract 키워드를 사용하여 선언
  - ✓ 인스턴스를 가질 수 없음
  - ✓ 접근 제한자를 명시하지 않으면 private 으로 선언됨

사용형식

```
abstract class 클래스이름
{
    // 클래스와 동일하게 구현
}
```

```
abstract class AbstractBase
{
    public abstract void SomeMethod(); // 추상 메소드 정의
}

class Derived : AbstractBase
{
    public override void SomeMethod()
    { // override 키워드 사용하여 메소드 재정의
        // Something
    }
}
```

- 추상 메소드
  - ✓ abstract 키워드 사용하여 메소드 선언
  - ✓ 파생클래스에서 override 키워드를 사용하여 반드시 구현되어야 됨
  - ✓ public, protected, internal, protected internal 접근 제한자 중 하나로 수식

추상 클래스 예제 코드

```
abstract class AbstractBase
{
    protected void PrivateMethod()
    {
        Console.WriteLine("AbstractBase.PrivateMethod()");
    }

    public void PublicMethodA()
    {
        Console.WriteLine("AbstractBase.PublicMethodA()");
    }

    public abstract void AbstractMethodA(); // 추상 메소드
}

class Derived : AbstractBase
{
    public override void AbstractMethodA()
    { // 추상 메소드 재정의
        Console.WriteLine("Derived.AbstractMethodA()");
        PrivateMethod();
    }
}
```

```
static void Main(string[] args)
{
    AbstractBase obj = new Derived();
    obj.AbstractMethodA();
    obj.PublicMethodA();
}
```

출력 결과:

```
Derived.AbstractMethodA()
AbstractBase.PrivateMethod()
AbstractBase.PublicMethodA()
```

추상 클래스의 프로퍼티

- 추상 프로퍼티 (Abstract Property)
  - ✓ abstract 키워드를 사용하여 선언
  - ✓ 파생 클래스에서 해당 프로퍼티 구현하도록 강제
  - ✓ 인터페이스의 프로퍼티와 유사

사용형식:

```
abstract class 추상클래스이름
{
    abstract 데이터형식 프로퍼티이름
    {
        get;
        set;
    }
}
```

사용예제

```
abstract class Product
{
    private static int serial = 0;
    public string SerialID
    { // 구현이 있는 프로퍼티
        get { return String.Format("{0:d5}", serial++); }
    }

    abstract public DateTime ProductDate
    { // 구현이 없는 추상 프로퍼티
        get; set;
    }
}

class MyProduct : Product
{
    public override DateTime ProductDate { get; set; }
}
```

추상 클래스의 프로퍼티 예제 코드

```
abstract class Product
{
    private static int serial = 0;
    public string SerialID
    { // 구현이 있는 프로퍼티
        get { return String.Format("{0:d5}", serial++); }
    }

    abstract public DateTime ProductDate
    { // 구현이 없는 추상 프로퍼티
        get; set;
    }
}

class MyProduct : Product
{
    public override DateTime ProductDate { get; set; }
}
```

```
static void Main(string[] args)
{
    Product product_1 = new MyProduct()
    { ProductDate = new DateTime(2018, 1, 10) };

    Console.WriteLine("Product: {0}, Product Date: {1}",
        product_1.SerialID,
        product_1.ProductDate);

    Product product_2 = new MyProduct()
    { ProductDate = new DateTime(2018, 2, 3) };

    Console.WriteLine("Product: {0}, Product Date: {1}",
        product_2.SerialID,
        product_2.ProductDate);
}
```

출력 결과:

```
Product: 00000, Product Date: 2018-01-10 오전 12:00:00
Product: 00001, Product Date: 2018-02-03 오전 12:00:00
```

SEJONG UNIVERSITY

파일과 디렉터리 정보 관리

- 파일은 컴퓨터 저장 매체에 기록되는 데이터의 묶음
- 디렉터리는 파일이 위치하는 주소이고 파일을 담는 폴더라고 불림
- .NET은 System.IO 네임스페이스 안에 다음의 클래스를 제공

클래스	설명
File	파일의 생성, 복사, 삭제, 이동, 조회를 처리하는 정적 메소드를 제공
FileInfo	File 클래스와 하는 일은 동일하지만 정적 메소드 대신 인스턴스 메소드를 제공
Directory	디렉터리의 생성, 삭제, 이동, 조회를 처리하는 정적 메소드를 제공
DirectoryInfo	DirectoryInfo 클래스와 하는 일은 동일하지만 정적 메소드 대신 인스턴스 메소드를 제공

주요 메소드와 프로퍼티

- File, FileInfo, Directory, DirectoryInfo 클래스가 제공하는 메소드와 프로퍼티
  - ✓ 파일/디렉터리의 생성, 복사, 삭제, 이동, 정보 조회 등의 기능을 수행

기능	File	FileInfo	Directory	DirectoryInfo
생성	Create( )	Create( )	CreateDirectory( )	Create( )
복사	Copy( )	CopyTo( )	-	-
삭제	Delete( )	Delete( )	Delete( )	Delete( )
이동	Move( )	MoveTo( )	Move( )	MoveTo( )
존재 여부 확인	Exists( )	Exists	Exists( )	Exists
속성 조회	GetAttributes( )	Attributes	GetAttributes( )	Attributes
하위 디렉터리 조회	-	-	GetDirectories( )	GetDirectories( )
하위 파일 조회	-	-	GetFiles( )	GetFiles( )

디렉터리/파일 정보조회 예제 코드

```
static void Main(string[] args)
{
    string directory;
    if (args.Length < 1)
        directory = ".";
    else
        directory = args[0];

    Console.WriteLine("{directory} directory Info");
    Console.WriteLine("- Directories : ");



    // 하위 디렉터리 목록 조회
    var directories = (from dir in Directory.GetDirectories(directory)
                       let info = new DirectoryInfo(dir)
                       select new
                       {
                           Name = info.Name,
                           Attributes = info.Attributes
                       }).ToList();

    foreach (var d in directories)
        Console.WriteLine($"{d.Name} : {d.Attributes}");
```

```
    // 하위 파일 목록 조회
    Console.WriteLine("- Files : ");
    var files = (from file in Directory.GetFiles(directory)
                 let info = new FileInfo(file)
                 select new
                 {
                     Name = info.Name,
                     FileSize = info.Length,
                     Attributes = info.Attributes
                 }).ToList();

    foreach (var f in files)
        Console.WriteLine($"{f.Name} : {f.FileSize},
{f.Attributes}");
}
```

출력 결과:

```
C:₩Users₩ongte₩source₩repos₩AbstractClass₩Dir₩bin₩Debug₩net5.0>Dir.exe "c:₩Users"
{directory} directory Info
- Directories :
All Users : Hidden, System, Directory, ReparsePoint, NotContentIndexed
Default : ReadOnly, Hidden, Directory
Default User : Hidden, System, Directory, ReparsePoint, NotContentIndexed
defaultuser100001 : Directory
ongte : Directory
Public : ReadOnly, Directory
- Files :
desktop.ini : 174, Hidden, System, Archive
```

System.IO.Stream 클래스

- 스트림 (Stream)
  - ✓ 데이터가 흐르는 통로를 의미
  - ✓ 저장매체(하드디스크)와 메모리 사이의 스트림을 놓은 후 파일에 담긴 데이터를 바이트 단위로 메모리로 차례차례 옮김
  - ✓ 파일을 처음부터 끝까지 순서대로 읽고 쓰는 "순차접근" 가능
  - ✓ 또한, 곧장 원하는 주소에 위치한 데이터에 접근하는 "임의접근" 가능
- .NET 에서 제공하는 Stream 클래스
  - ✓ 추상 클래스 형태로 인스턴스를 만들기 위해 파생클래스를 이용
  - ✓ 하나의 스트림 모델로 다양한 매체나 장치들에 대한 파일 입출력을 다룰 수 있음
  - ✓ 주요 파생클래스
    - FileStream : 디스크 파일에 데이터를 기록
    - NetworkStream : 네트워크를 통해 데이터를 주고받도록 지원
    - BufferdStream : 데이터를 메모리 버퍼에 담았다가 일정량이 쌓일 때마다 기록

FileStream 클래스 : 파일 쓰기

- FileStream 클래스의 인스턴스 생성

```
Stream stream1 = new FileStream("a.dat", FileMode.Create); // 새 파일 생성
Stream stream2 = new FileStream("b.dat", FileMode.Open);   // 파일 열기
Stream stream3 = new FileStream("c.dat", FileMode.OpenOrCreate);  // 파일을 열거나
                                                                  // 파일이 없으면 생성
Stream stream4 = new FileStream("d.dat", FileMode.Truncate);  // 파일을 비워서 열기
Stream stream5 = new FileStream("e.dat", FileMode.Append);    // 덧붙이기 모드로 열기
```

- 파일에 데이터를 기록하기 위해 Stream 클래스로 부터 물려받은 메소드
  - ✓ Write( )와 WriteByte( ) 메소드는 매개변수로 byte 또는 byte 배열만 입력 받음

```
public override void Write(
    byte[] array, // 쓸 데이터가 담겨 있는 byte 배열
    int offset,   // byte 배열 내의 시작 오프셋
    int count     // 기록할 데이터의 총 길이(단위는 바이트)
    );

public override void WriteByte(byte value);
```

FileStream 클래스 : 파일 쓰기

- BitConverter 클래스
  - ✓ 임의 형식의 데이터를 byte의 배열로 변환
  - ✓ byte의 배열에 담겨있는 데이터를 다시 임의 형식으로 변환
  - ✓ 예를 들어, long 형식의 데이터를 파일에 기록

```
long someValue = 0x123456789ABCDEF0;

//  1) 파일 스트림 생성
Stream outStream = new FileStream("a.dat", FileMode.Create);

// 2) someValue(long 형식)를 byte 배열로 변환
byte[] wBytes = BitConverter.GetBytes(someValue);

// 3) 변환한 byte 배열을 파일 스트림을 통해 파일에 기록
outStream.Write(wBytes, 0, wBytes.Length);

// 4) 파일 스트림 닫기
outStream.Close();
```

FileStream 클래스 : 파일 읽기

- 파일에서 데이터를 읽기 위해 Stream 클래스로 부터 물려받은 두 개의 메소드

메소드 정의:

```
public override int Read(
    byte[] array, // 읽은 데이터를 담을 byte 배열
    int offset,   // byte 배열 내의 시작 오프셋
    int count     // 읽을 데이터의 최대 바이트 수
    );

public override int ReadByte();
```

메소드 사용예제 (long형 데이터를 파일에서 읽기):

```
byte[] rBytes = new byte[8];

// 1) 파일 스트림 생성
Stream inStream = new FileStream("a.dat",
FileMode.Open);

// 2) rBytes의 길이만큼(8바이트) 데이터를 읽어 rBytes에
저장
inStream.Read(rBytes, 0, rBytes.Length);

// 3) BitConverter를 이용하여 rBytes에 담겨 있는 값을
long 형식으로 변환
long readValue = BitConverter.ToInt64(rBytes, 0);

// 4) 파일 스트림 닫기
inStream.Close();
```

FileStream 클래스 예제코드

```
static void Main(string[] args)
{
    long someValue = 0x123456789ABCDEF0;
    // X16에서 X는 수를 16진수로 표현하고 뒤의 숫자 16은 열여섯 자리 수로 표현
    Console.WriteLine("{0,-13} : 0x{1:X16}", "Original Data", someValue);

    Stream outStream = new FileStream("a.dat", FileMode.Create);
    byte[] wBytes = BitConverter.GetBytes(someValue);
    Console.Write("{0,-13} : ", "Byte array");

    foreach (byte b in wBytes)
        Console.Write("{0:X2} ", b);
    Console.WriteLine();

    outStream.Write(wBytes, 0, wBytes.Length);
    outStream.Close();

    byte[] rBytes = new byte[8];

    Stream inStream = new FileStream("a.dat", FileMode.Open);
    inStream.Read(rBytes, 0, rBytes.Length);

    long readValue = BitConverter.ToInt64(rBytes, 0);
    Console.WriteLine("{0,-13} : 0x{1:X16}", "Read Data", readValue);
    inStream.Close();
}
```

C#에서 지원하는 바이트 오더가
데이터의 낮은 주소부터
기록하는 리틀 엔디안 방식

출력 결과:

```
Original Data : 0x123456789ABCDEF0
Byte array    : F0 DE BC 9A 78 56 34 12
Read Data     : 0x123456789ABCDEF0
```

Thank you!